

Southern and Volga Russian Regional Contest

Saratov State University

28 октября 2025 г.

A. Delete the Array

Удаляем либо два первых, если они равны, либо любой из минимумов.

A. Delete the Array

Удаляем либо два первых, если они равны, либо любой из минимумов.

- Жадные идеи

A. Delete the Array

Удаляем либо два первых, если они равны, либо любой из минимумов.

- Жадные идеи
- Выгодно в первую очередь удалять пары равных.

A. Delete the Array

Удаляем либо два первых, если они равны, либо любой из минимумов.

- Жадные идеи
- Выгодно в первую очередь удалять пары равных.
- Из минимумов выгодно удалять самый левый.

A. Delete the Array

Удаляем либо два первых, если они равны, либо любой из минимумов.

- Жадные идеи
- Выгодно в первую очередь удалять пары равных.
- Из минимумов выгодно удалять самый левый.

Сложность:

Научиться моделировать удаления достаточно быстро.

A. Delete the Array

Как моделировать:

A. Delete the Array

Как моделировать:

- Честно удалять элементы из массива (`list`, `vector`) — медленно; порядка $O(n^2)$ времени.

A. Delete the Array

Как моделировать:

- Честно удалять элементы из массива (`list`, `vector`) — медленно; порядка $O(n^2)$ времени.
- Нам понадобятся:
 - Список индексов i в порядке возрастания a_i (*ord*) — в этом порядке мы их и будем удалять.
 - Sorted set (`set` в C++) индексов i — индексы еще не удаленных на данный момент элементов в порядке возрастания.

A. Delete the Array

Как моделировать:

- Честно удалять элементы из массива (`list`, `vector`) — медленно; порядка $O(n^2)$ времени.
- Нам понадобятся:
 - Список индексов i в порядке возрастания a_i (ord) — в этом порядке мы их и будем удалять.
 - Sorted set (`set` в C++) индексов i — индексы еще не удаленных на данный момент элементов в порядке возрастания.
- Удаляем из сета элементы в порядке ord .
- Перед удалением очередного элемента удаляем как можно больше равных пар (`set.begin()` и `next(set.begin())`).

A. Delete the Array

Как моделировать:

- Честно удалять элементы из массива (`list`, `vector`) — медленно; порядка $O(n^2)$ времени.
- Нам понадобятся:
 - Список индексов i в порядке возрастания a_i (ord) — в этом порядке мы их и будем удалять.
 - Sorted set (`set` в C++) индексов i — индексы еще не удаленных на данный момент элементов в порядке возрастания.
- Удаляем из сета элементы в порядке ord .
- Перед удалением очередного элемента удаляем как можно больше равных пар (`set.begin()` и `next(set.begin())`).
- Порядка $O(n \log n)$ времени.

A. Delete the Array

Как моделировать:

- Честно удалять элементы из массива (`list`, `vector`) — медленно; порядка $O(n^2)$ времени.
- Нам понадобятся:
 - Список индексов i в порядке возрастания a_i (ord) — в этом порядке мы их и будем удалять.
 - Sorted set (`set` в C++) индексов i — индексы еще не удаленных на данный момент элементов в порядке возрастания.
- Удаляем из сета элементы в порядке ord .
- Перед удалением очередного элемента удаляем как можно больше равных пар (`set.begin()` и `next(set.begin())`).
- Порядка $O(n \log n)$ времени.

Можно за линейное время.

B. Convex Interval

В списке точек на плоскости найти самый длинный отрезок, образующий выпуклый многоугольник

B. Convex Interval

В списке точек на плоскости найти самый длинный отрезок, образующий выпуклый многоугольник

- Если удалить из выпуклого многоугольника любую точку, он все еще будет выпуклым

B. Convex Interval

В списке точек на плоскости найти самый длинный отрезок, образующий выпуклый многоугольник

- Если удалить из выпуклого многоугольника любую точку, он все еще будет выпуклым
- Если $[l, r]$ выпуклый, то и $[l + 1, r]$, и $[l, r - 1]$ выпуклые

B. Convex Interval

В списке точек на плоскости найти самый длинный отрезок, образующий выпуклый многоугольник

- Если удалить из выпуклого многоугольника любую точку, он все еще будет выпуклым
- Если $[l, r]$ выпуклый, то и $[l + 1, r]$, и $[l, r - 1]$ выпуклые
- Значит можно писать два указателя

B. Convex Interval

- Как проверить, что $[l, r + 1]$ выпуклый, если известно, что $[l, r]$ выпуклый

B. Convex Interval

- Как проверить, что $[l, r + 1]$ выпуклый, если известно, что $[l, r]$ выпуклый
- Оказывается, достаточно проверить, что тройки точек, включающие точку $r + 1$, расположены по часовой стрелке

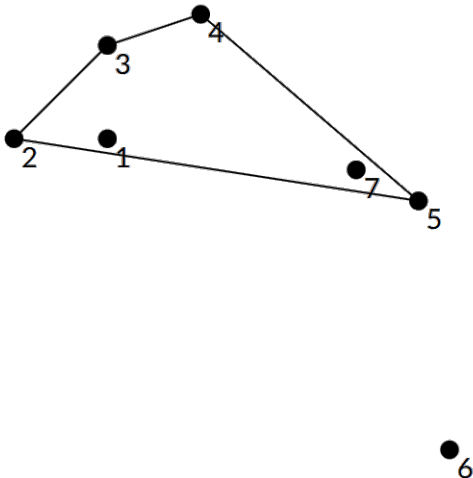
B. Convex Interval

- Как проверить, что $[l, r + 1]$ выпуклый, если известно, что $[l, r]$ выпуклый
- Оказывается, достаточно проверить, что тройки точек, включающие точку $r + 1$, расположены по часовой стрелке
- Это можно сделать с помощью псевдоскалярного произведения

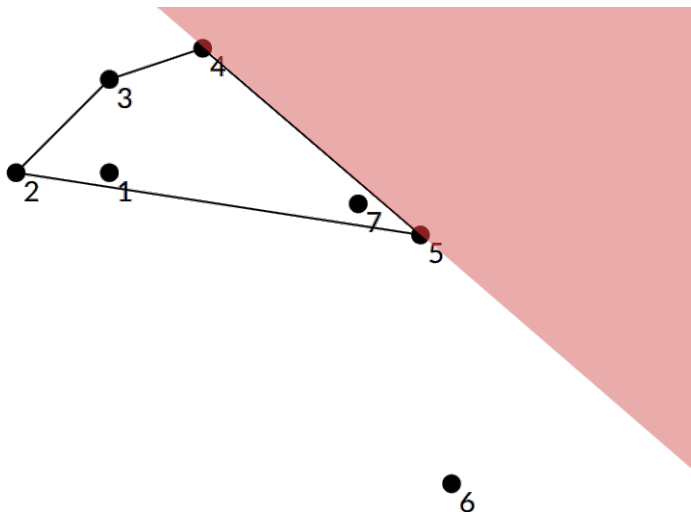
B. Convex Interval

- Как проверить, что $[l, r + 1]$ выпуклый, если известно, что $[l, r]$ выпуклый
- Оказывается, достаточно проверить, что тройки точек, включающие точку $r + 1$, расположены по часовой стрелке
- Это можно сделать с помощью псевдоскалярного произведения
- Рассмотрим, где должна находиться новая точка, чтобы многоугольник остался выпуклым

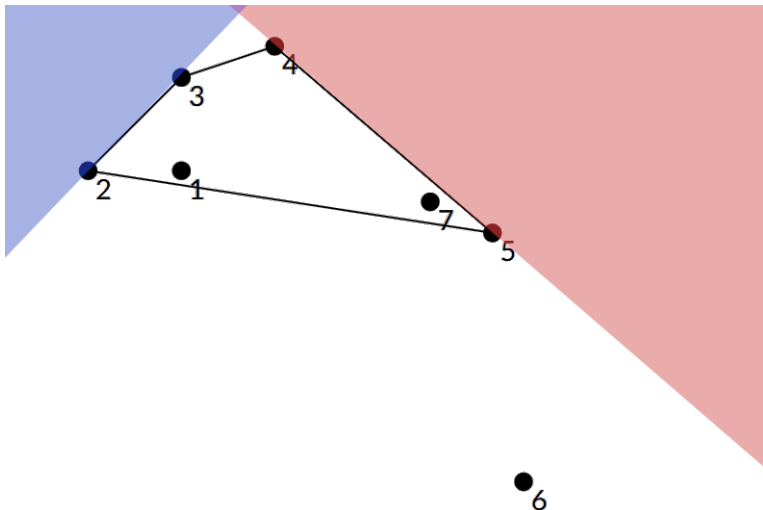
B. Convex Interval



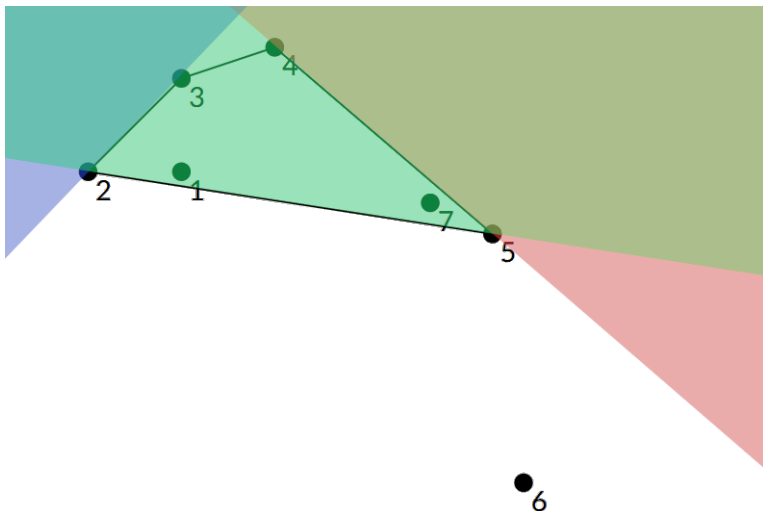
B. Convex Interval



B. Convex Interval



B. Convex Interval



C. Monocarp, Polycarp and Brackets

Два игрока играют в игру. У них есть строка из скобок s ; у первого игрока есть пустая строка t . Каждый игрок на своем ходу удаляет первый или последний символ s ; первый игрок при этом приписывает его к своей строке. Первый игрок выигрывает, если в какой-то момент t — правильная скобочная последовательность. Определить, кто выигрывает

C. Monocarp, Polycarp and Brackets

Два игрока играют в игру. У них есть строка из скобок s ; у первого игрока есть пустая строка t . Каждый игрок на своем ходу удаляет первый или последний символ s ; первый игрок при этом приписывает его к своей строке. Первый игрок выигрывает, если в какой-то момент t — правильная скобочная последовательность. Определить, кто выиграет

- Строка t либо на каком-то префиксе уже сломалась, либо еще нет, тогда нас интересует только ее баланс

C. Monocarp, Polycarp and Brackets

Два игрока играют в игру. У них есть строка из скобок s ; у первого игрока есть пустая строка t . Каждый игрок на своем ходу удаляет первый или последний символ s ; первый игрок при этом приписывает его к своей строке. Первый игрок выигрывает, если в какой-то момент t — правильная скобочная последовательность. Определить, кто выиграет

- Строка t либо на каком-то префиксе уже сломалась, либо еще нет, тогда нас интересует только ее баланс
- Если он станет 0, первый игрок выигрывает

C. Monocarp, Polycarp and Brackets

Два игрока играют в игру. У них есть строка из скобок s ; у первого игрока есть пустая строка t . Каждый игрок на своем ходу удаляет первый или последний символ s ; первый игрок при этом приписывает его к своей строке. Первый игрок выигрывает, если в какой-то момент t — правильная скобочная последовательность. Определить, кто выиграет

- Строка t либо на каком-то префиксе уже сломалась, либо еще нет, тогда нас интересует только ее баланс
- Если он станет 0, первый игрок выигрывает
- Состояние игры: кол-во символов, взятых слева и справа, и баланс t

C. Monocarp, Polycarp and Brackets

Два игрока играют в игру. У них есть строка из скобок s ; у первого игрока есть пустая строка t . Каждый игрок на своем ходу удаляет первый или последний символ s ; первый игрок при этом приписывает его к своей строке. Первый игрок выигрывает, если в какой-то момент t — правильная скобочная последовательность. Определить, кто выиграет

- Строка t либо на каком-то префиксе уже сломалась, либо еще нет, тогда нас интересует только ее баланс
- Если он станет 0, первый игрок выигрывает
- Состояние игры: кол-во символов, взятых слева и справа, и баланс t
- Через кол-во удаленных символов легко понять, чей ход
- Динамика: $dp_{l,r,x}$ — кто выиграет, если удалено l символов слева и r справа, а баланс строки t равен x

C. Monocarp, Polycarp and Brackets

Это работает за $O(n^3)$, как ускорять?

C. Monocarp, Polycarp and Brackets

Это работает за $O(n^3)$, как ускорять?

- Если баланс был x и стал $y < x$, то все числа $[y, x]$ были в качестве баланса

C. Monocarp, Polycarp and Brackets

Это работает за $O(n^3)$, как ускорять?

- Если баланс был x и стал $y < x$, то все числа $[y, x]$ были в качестве баланса
- Поэтому чем меньше баланс, тем лучше для первого игрока

C. Monocarp, Polycarp and Brackets

Это работает за $O(n^3)$, как ускорять?

- Если баланс был x и стал $y < x$, то все числа $[y, x]$ были в качестве баланса
- Поэтому чем меньше баланс, тем лучше для первого игрока
- $dp'_{l,r}$ — максимальный баланс, при котором выигрывает первый игрок, если удалено l символов слева и r справа

C. Monocarp, Polycarp and Brackets

Это работает за $O(n^3)$, как ускорять?

- Если баланс был x и стал $y < x$, то все числа $[y, x]$ были в качестве баланса
- Поэтому чем меньше баланс, тем лучше для первого игрока
- $dp'_{l,r}$ — максимальный баланс, при котором выигрывает первый игрок, если удалено l символов слева и r справа
- Можно считать в порядке убывания $l + r$, чтобы оставшийся «кусок» строки увеличивался

C. Monocarp, Polycarp and Brackets

Это работает за $O(n^3)$, как ускорять?

- Если баланс был x и стал $y < x$, то все числа $[y, x]$ были в качестве баланса
- Поэтому чем меньше баланс, тем лучше для первого игрока
- $dp'_{l,r}$ — максимальный баланс, при котором выигрывает первый игрок, если удалено l символов слева и r справа
- Можно считать в порядке убывания $l + r$, чтобы оставшийся «кусок» строки увеличивался
- Первый игрок выигрывает, если $dp'_{1,0} \geq 1$ или $d'_{0,1} \geq 1$, и можно сделать валидный ход

По данной информации о вкусоности и объеме крыжовника в каждый день, выбрать дни покупок, чтобы максимизировать минимальное удовольствие по скользящим окнам из m дней

По данной информации о вкусоности и объеме крыжовника в каждый день, выбрать дни покупок, чтобы максимизировать минимальное удовольствие по скользящим окнам из m дней

- Просят минимизировать максимум $\rightarrow$ бинарный поиск

По данной информации о вкусоности и объеме крыжовника в каждый день, выбрать дни покупок, чтобы максимизировать минимальное удовольствие по скользящим окнам из m дней

- Просят минимизировать максимум $\rightarrow$ бинарный поиск
- Проанализируем потенциальные минимальные окна:

По данной информации о вкусности и объеме крыжовника в каждый день, выбрать дни покупок, чтобы максимизировать минимальное удовольствие по скользящим окнам из m дней

- Просят минимизировать максимум $\rightarrow$ бинарный поиск
- Проанализируем потенциальные минимальные окна:
- $b_i \cdot m$ внутри каждого купленного отрезка

По данной информации о вкусоности и объеме крыжовника в каждый день, выбрать дни покупок, чтобы максимизировать минимальное удовольствие по скользящим окнам из m дней

- Просят минимизировать максимум $\rightarrow$ бинарный поиск
- Проанализируем потенциальные минимальные окна:
- $b_i \cdot m$ внутри каждого купленного отрезка
- Если между двумя соседними купленными отрезками есть пустые дни, то они точно приносят меньше удовольствия, чем внутри обоих соседей

По данной информации о вкусности и объеме крыжовника в каждый день, выбрать дни покупок, чтобы максимизировать минимальное удовольствие по скользящим окнам из m дней

- Просят минимизировать максимум $\rightarrow$ бинарный поиск
- Проанализируем потенциальные минимальные окна:
- $b_i \cdot m$ внутри каждого купленного отрезка
- Если между двумя соседними купленными отрезками есть пустые дни, то они точно приносят меньше удовольствия, чем внутри обоих соседей
- Если есть окно, включающее два соседних отрезка, то это окно можно подвинуть до границы одного из отрезков

По данной информации о вкусности и объеме крыжовника в каждый день, выбрать дни покупок, чтобы максимизировать минимальное удовольствие по скользящим окнам из m дней

- Просят минимизировать максимум $\rightarrow$ бинарный поиск
- Проанализируем потенциальные минимальные окна:
- $b_i \cdot m$ внутри каждого купленного отрезка
- Если между двумя соседними купленными отрезками есть пустые дни, то они точно приносят меньше удовольствия, чем внутри обоих соседей
- Если есть окно, включающее два соседних отрезка, то это окно можно подвинуть до границы одного из отрезков
- Если правый из отрезков отсекается n -м днем, то возможно не получится подвинуть окно

D. Gooseberry

- Хотелось бы писать динамику вида dp_i — можно ли купить i -й отрезок, а также некоторые до него, чтобы гарантировать удовольствие $\geq mid$

D. Gooseberry

- Хотелось бы писать динамику вида dp_i — можно ли купить i -й отрезок, а также некоторые до него, чтобы гарантировать удовольствие $\geq mid$
- dp_i будет истинно, если найдется такой отрезок, который заканчивается слева от i -го, а отрезок пустых дней между ними не мешает условию бинарного поиска

D. Gooseberry

- Хотелось бы писать динамику вида dp_i — можно ли купить i -й отрезок, а также некоторые до него, чтобы гарантировать удовольствие $\geq mid$
- dp_i будет истинно, если найдется такой отрезок, который заканчивается слева от i -го, а отрезок пустых дней между ними не мешает условию бинарного поиска
- Заметим, что проверка условия выполняется независимо для двух соседних отрезков

- Хотелось бы писать динамику вида dp_i — можно ли купить i -й отрезок, а также некоторые до него, чтобы гарантировать удовольствие $\geq mid$
- dp_i будет истинно, если найдется такой отрезок, который заканчивается слева от i -го, а отрезок пустых дней между ними не мешает условию бинарного поиска
- Заметим, что проверка условия выполняется независимо для двух соседних отрезков
- Отрезок может позволить себе $m - \lceil \frac{mid}{b_i} \rceil$ пустых дней рядом с собой

- Хотелось бы писать динамику вида dp_i — можно ли купить i -й отрезок, а также некоторые до него, чтобы гарантировать удовольствие $\geq mid$
- dp_i будет истинно, если найдется такой отрезок, который заканчивается слева от i -го, а отрезок пустых дней между ними не мешает условию бинарного поиска
- Заметим, что проверка условия выполняется независимо для двух соседних отрезков
- Отрезок может позволить себе $m - \lceil \frac{mid}{b_i} \rceil$ пустых дней рядом с собой
- Напишем сканлайн и будем поддерживать закрытые отрезки в сете по возрастанию $i + a_i$

- Хотелось бы писать динамику вида dp_i — можно ли купить i -й отрезок, а также некоторые до него, чтобы гарантировать удовольствие $\geq mid$
- dp_i будет истинно, если найдется такой отрезок, который заканчивается слева от i -го, а отрезок пустых дней между ними не мешает условию бинарного поиска
- Заметим, что проверка условия выполняется независимо для двух соседних отрезков
- Отрезок может позволить себе $m - \lceil \frac{mid}{b_i} \rceil$ пустых дней рядом с собой
- Напишем сканлайн и будем поддерживать закрытые отрезки в сете по возрастанию $i + a_i$
- Отрезок добавляется в сет в момент $i + a_i$, а удаляется — в $i + a_i + m - \lceil \frac{mid}{b_i} \rceil$

- Оказывается, последний отрезок, который вылезает за n , работает не сильно по-другому

- Оказывается, последний отрезок, который вылезает за n , работает не сильно по-другому
- Достаточно дополнительно проверить последнее окно длины m

- Оказывается, последний отрезок, который вылезает за n , работает не сильно по-другому
- Достаточно дополнительно проверить последнее окно длины m
- В это окно попадает весь текущий отрезок, а также возможно прошлый

- Оказывается, последний отрезок, который вылезает за n , работает не сильно по-другому
- Достаточно дополнительно проверить последнее окно длины m
- В это окно попадает весь текущий отрезок, а также возможно прошлый
- Хотелось бы найти отрезок, который заканчивается после дня $n - m$, с максимальной суммой

- Оказывается, последний отрезок, который вылезает за n , работает не сильно по-другому
- Достаточно дополнительно проверить последнее окно длины m
- В это окно попадает весь текущий отрезок, а также возможно прошлый
- Хотелось бы найти отрезок, который заканчивается после дня $n - m$, с максимальной суммой
- Достаточно ли хранить второй сет с теми же событиями, только отсортированный по сумме удовольствий после дня $n - m$?

- Оказывается, достаточно. Разберем случаи)

- Оказывается, достаточно. Разберем случаи)
- Если у последнего отрезка b меньше, то нам достаточно, что предпоследний дотягивается

- Оказывается, достаточно. Разберем случаи)
- Если у последнего отрезка b меньше, то нам достаточно, что предпоследний дотягивается
- Если у последнего отрезка b больше, то его покупка точно не обновит минимум

- Оказывается, достаточно. Разберем случаи)
- Если у последнего отрезка b меньше, то нам достаточно, что предпоследний дотягивается
- Если у последнего отрезка b больше, то его покупка точно не обновит минимум
- Для восстановления ответа можно в сетах поддерживать и индексы отрезков

- Оказывается, достаточно. Разберем случаи)
- Если у последнего отрезка b меньше, то нам достаточно, что предпоследний дотягивается
- Если у последнего отрезка b больше, то его покупка точно не обновит минимум
- Для восстановления ответа можно в сетях поддерживать и индексы отрезков
- Асимптотика решения: $O(n \log A \log n)$

Сообщить, насколько далеко может доехать $1 \times k$ машина на поле с некоторыми занятыми клетками

Сообщить, насколько далеко может доехать $1 \times k$ машина на поле с некоторыми занятыми клетками

- Построим граф передвижений машины: из клетки (x, y) в клетки $(x, y - 1)$, $(x, y + 1)$ и $(x + 1, y)$

Сообщить, насколько далеко может доехать $1 \times k$ машина на поле с некоторыми занятыми клетками

- Построим граф передвижений машины: из клетки (x, y) в клетки $(x, y - 1)$, $(x, y + 1)$ и $(x + 1, y)$
- Чтобы проверить, что машина может стоять в клетке (x, y) , найдем расстояние до ближайшей занятой клетки снизу

Сообщить, насколько далеко может доехать $1 \times k$ машина на поле с некоторыми занятыми клетками

- Построим граф передвижений машины: из клетки (x, y) в клетки $(x, y - 1)$, $(x, y + 1)$ и $(x + 1, y)$
- Чтобы проверить, что машина может стоять в клетке (x, y) , найдем расстояние до ближайшей занятой клетки снизу

$$nxt_{x,y} = \begin{cases} nxt_{x+1,y} + 1, & \text{if } (x, y) \text{ is free} \\ 0, & \text{otherwise} \end{cases}$$

- Неожиданный момент: обход в глубину получает RE из-за размера стека

- Неожиданный момент: обход в глубину получает RE из-за размера стека
- Напишем обход в ширину

Нужно узнать, кто выиграет в игру Дженга при оптимальной стратегии обоих игроков.

Нужно узнать, кто выиграет в игру Дженга при оптимальной стратегии обоих игроков.

Существует 3 типа рядов:

Нужно узнать, кто выиграет в игру Дженга при оптимальной стратегии обоих игроков.

Существует 3 типа рядов:

- из которых нельзя удалить ни одного блока (тип 0);

Нужно узнать, кто выиграет в игру Дженга при оптимальной стратегии обоих игроков.

Существует 3 типа рядов:

- из которых нельзя удалить ни одного блока (тип 0);
- из которых можно удалить ровно 1 блок (тип 1);

Нужно узнать, кто выиграет в игру Дженга при оптимальной стратегии обоих игроков.

Существует 3 типа рядов:

- из которых нельзя удалить ни одного блока (тип 0);
- из которых можно удалить ровно 1 блок (тип 1);
- из которых можно удалить 1 или 2 блока, в зависимости от хода (тип 2).

Нужно узнать, кто выиграет в игру Дженга при оптимальной стратегии обоих игроков.

Существует 3 типа рядов:

- из которых нельзя удалить ни одного блока (тип 0);
- из которых можно удалить ровно 1 блок (тип 1);
- из которых можно удалить 1 или 2 блока, в зависимости от хода (тип 2).

Возможные решения.

Нужно узнать, кто выиграет в игру Дженга при оптимальной стратегии обоих игроков.

Существует 3 типа рядов:

- из которых нельзя удалить ни одного блока (тип 0);
- из которых можно удалить ровно 1 блок (тип 1);
- из которых можно удалить 1 или 2 блока, в зависимости от хода (тип 2).

Возможные решения.

- Строгое решение — теория Шпрага-Гранди.

Нужно узнать, кто выиграет в игру Дженга при оптимальной стратегии обоих игроков.

Существует 3 типа рядов:

- из которых нельзя удалить ни одного блока (тип 0);
- из которых можно удалить ровно 1 блок (тип 1);
- из которых можно удалить 1 или 2 блока, в зависимости от хода (тип 2).

Возможные решения.

- Строгое решение — теория Шпрага-Гранди.
- Конструктивное решение.

Ф. Јенга. Строгое решение.

Применим теорию Шпрага-Гранди.

Применим теорию Шпрага-Гранди. Каждому ряду сопоставляется игра Ним:

Применим теорию Шпрага-Гранди. Каждому ряду сопоставляется игра Ним:

- ряду типа 0 — Ним(0) с характеристической функцией $\chi_0 = 0$;

Применим теорию Шпрага-Гранди. Каждому ряду сопоставляется игра Ним:

- ряду типа 0 — Ним(0) с характеристической функцией $\chi_0 = 0$;
- ряду типа 1 — Ним(1) с характеристической функцией $\chi_1 = 1$;

Применим теорию Шпрага-Гранди. Каждому ряду сопоставляется игра Ним:

- ряду типа 0 — Ним(0) с характеристической функцией $\chi_0 = 0$;
- ряду типа 1 — Ним(1) с характеристической функцией $\chi_1 = 1$;
- ряду типа 2 — Ним(2) с характеристической функцией $\chi_2 = 2$.

Применим теорию Шпрага-Гранди. Каждому ряду сопоставляется игра Ним:

- ряду типа 0 — Ним(0) с характеристической функцией $\chi_0 = 0$;
- ряду типа 1 — Ним(1) с характеристической функцией $\chi_1 = 1$;
- ряду типа 2 — Ним(2) с характеристической функцией $\chi_2 = 2$.

Вычисляем характеристическую функцию χ исходной игры как XOR характеристических функций всех рядов.

Применим теорию Шпрага-Гранди. Каждому ряду сопоставляется игра Ним:

- ряду типа 0 — Ним(0) с характеристической функцией $\chi_0 = 0$;
- ряду типа 1 — Ним(1) с характеристической функцией $\chi_1 = 1$;
- ряду типа 2 — Ним(2) с характеристической функцией $\chi_2 = 2$.

Вычисляем характеристическую функцию χ исходной игры как XOR характеристических функций всех рядов.

- При $\chi \neq 0$ побеждает Монокарп.

Применим теорию Шпрага-Гранди. Каждому ряду сопоставляется игра Ним:

- ряду типа 0 — Ним(0) с характеристической функцией $\chi_0 = 0$;
- ряду типа 1 — Ним(1) с характеристической функцией $\chi_1 = 1$;
- ряду типа 2 — Ним(2) с характеристической функцией $\chi_2 = 2$.

Вычисляем характеристическую функцию χ исходной игры как XOR характеристических функций всех рядов.

- При $\chi \neq 0$ побеждает Монокарп.
- При $\chi = 0$ побеждает Поликарп.

F. Jenga. Конструктивное решение.

- Ряды типа 0 никак не влияют на игру.

F. Jenga. Конструктивное решение.

- Ряды типа 0 никак не влияют на игру.
- Если есть максимум по одному ряду типов 1 и 2 то:

F. Jenga. Конструктивное решение.

- Ряды типа 0 никак не влияют на игру.
- Если есть максимум по одному ряду типов 1 и 2 то:
 - при $c_1 = 0, c_2 = 0$ выигрывает Поликарп (у Монокарпа нет ходов);

F. Jenga. Конструктивное решение.

- Ряды типа 0 никак не влияют на игру.
- Если есть максимум по одному ряду типов 1 и 2 то:
 - при $c_1 = 0, c_2 = 0$ выигрывает Поликарп (у Монокарпа нет ходов);
 - при $c_1 = 1, c_2 = 0$ выигрывает Монокарп, превращая ряд типа 1 в ряд типа 0;

- Ряды типа 0 никак не влияют на игру.
- Если есть максимум по одному ряду типов 1 и 2 то:
 - при $c_1 = 0, c_2 = 0$ выигрывает Поликарп (у Монокарпа нет ходов);
 - при $c_1 = 1, c_2 = 0$ выигрывает Монокарп, превращая ряд типа 1 в ряд типа 0;
 - при $c_1 = 0, c_2 = 1$ выигрывает Монокарп, превращая ряд типа 2 в ряд типа 0;

- Ряды типа 0 никак не влияют на игру.
- Если есть максимум по одному ряду типов 1 и 2 то:
 - при $c_1 = 0, c_2 = 0$ выигрывает Поликарп (у Монокарпа нет ходов);
 - при $c_1 = 1, c_2 = 0$ выигрывает Монокарп, превращая ряд типа 1 в ряд типа 0;
 - при $c_1 = 0, c_2 = 1$ выигрывает Монокарп, превращая ряд типа 2 в ряд типа 0;
 - при $c_1 = 1, c_2 = 1$ выигрывает Монокарп, превращая ряд типа 2 в ряд типа 1.

- Ряды типа 0 никак не влияют на игру.
- Если есть максимум по одному ряду типов 1 и 2 то:
 - при $c_1 = 0, c_2 = 0$ выигрывает Поликарп (у Монокарпа нет ходов);
 - при $c_1 = 1, c_2 = 0$ выигрывает Монокарп, превращая ряд типа 1 в ряд типа 0;
 - при $c_1 = 0, c_2 = 1$ выигрывает Монокарп, превращая ряд типа 2 в ряд типа 0;
 - при $c_1 = 1, c_2 = 1$ выигрывает Монокарп, превращая ряд типа 2 в ряд типа 1.
- Добавление четного количества рядов как типа 1, так и типа 2 не меняет победителя, т.к. существует симметричная стратегия по добавленным рядам.

G. String Transformation

Есть строка. Мы проводим одну операцию: выбрать подотрезок и символ, удалить все вхождения выбранного символа с этого подотрезка. Посчитать, сколько различных строк может получиться

G. String Transformation

Есть строка. Мы проводим одну операцию: выбрать подотрезок и символ, удалить все вхождения выбранного символа с этого подотрезка. Посчитать, сколько различных строк может получиться

- Переберем символ, который удалим

G. String Transformation

Есть строка. Мы проводим одну операцию: выбрать подотрезок и символ, удалить все вхождения выбранного символа с этого подотрезка. Посчитать, сколько различных строк может получиться

- Переберем символ, который удалим
- Выделим непрерывные блоки вхождений этого символа
aaabcbabcaab, символ a: [3, 1, 2]

G. String Transformation

Есть строка. Мы проводим одну операцию: выбрать подотрезок и символ, удалить все вхождения выбранного символа с этого подотрезка. Посчитать, сколько различных строк может получиться

- Переберем символ, который удалим
- Выделим непрерывные блоки вхождений этого символа
aaabcbabcaab, символ a: [3, 1, 2]
- Удаляемые символы либо все в одном блоке, либо в нескольких

G. String Transformation

Есть строка. Мы проводим одну операцию: выбрать подотрезок и символ, удалить все вхождения выбранного символа с этого подотрезка. Посчитать, сколько различных строк может получиться

- Переберем символ, который удалим
- Выделим непрерывные блоки вхождений этого символа
aaabcbabcaab, символ a: [3, 1, 2]
- Удаляемые символы либо все в одном блоке, либо в нескольких
- Если в нескольких и правая граница в каком-то блоке, левая граница может быть в любом блоке левее

G. String Transformation

Есть строка. Мы проводим одну операцию: выбрать подотрезок и символ, удалить все вхождения выбранного символа с этого подотрезка. Посчитать, сколько различных строк может получиться

- Переберем символ, который удалим
- Выделим непрерывные блоки вхождений этого символа
aaabcsabcsaab, символ a: [3, 1, 2]
- Удаляемые символы либо все в одном блоке, либо в нескольких
- Если в нескольких и правая граница в каком-то блоке, левая граница может быть в любом блоке левее
- Если в одном блоке, можно перебрать блок и сколько символов из него удалим

G. String Transformation

Есть строка. Мы проводим одну операцию: выбрать подотрезок и символ, удалить все вхождения выбранного символа с этого подотрезка. Посчитать, сколько различных строк может получиться

- Переберем символ, который удалим
- Выделим непрерывные блоки вхождений этого символа
aaabcbabcaab, символ a: [3, 1, 2]
- Удаляемые символы либо все в одном блоке, либо в нескольких
- Если в нескольких и правая граница в каком-то блоке, левая граница может быть в любом блоке левее
- Если в одном блоке, можно перебрать блок и сколько символов из него удалим
- Все строки, полученные таким образом, будут различными

H. Rigged Matchmaking

Нужно построить максимальное количество пар $(l_M, z) - (x, y)$, где

- все $z \in [l_M + 1, r_M]$ и все различные;
- все $x, y \in [l_B, r_B]$ и все различные;
- $l_M + z \geq x + y$ во всех парах.

H. Rigged Matchmaking

Строим пары $(l_M, z) - (x, y)$:

H. Rigged Matchmaking

Строим пары $(I_M, z) - (x, y)$:

- Бинарный поиск по ответу.

H. Rigged Matchmaking

Строим пары $(l_M, z) - (x, y)$:

- Бинарный поиск по ответу.
- k «лучших» союзников, т. е. $(r_M - k + 1), (r_M - k + 2), \dots, r_M$.
- $2k$ «лучших» противников, т. е. $l_B, (l_B + 1), \dots, (l_B + 2k - 1)$.

H. Rigged Matchmaking

Строим пары $(l_M, z) - (x, y)$:

- Бинарный поиск по ответу.
- k «лучших» союзников, т. е. $(r_M - k + 1), (r_M - k + 2), \dots, r_M$.
- $2k$ «лучших» противников, т. е. $l_B, (l_B + 1), \dots, (l_B + 2k - 1)$.

Нужно проверить, можно ли как-то попарить.

H. Rigged Matchmaking

Строим пары $(l_M, z) - (x, y)$:

- Бинарный поиск по ответу.
- k «лучших» союзников, т. е. $(r_M - k + 1), (r_M - k + 2), \dots, r_M$.
- $2k$ «лучших» противников, т. е. $l_B, (l_B + 1), \dots, (l_B + 2k - 1)$.

Нужно проверить, можно ли как-то попарить.

- Большие ограничения $\Rightarrow$ что-то простое.

H. Rigged Matchmaking

Строим пары $(l_M, z) - (x, y)$:

- Бинарный поиск по ответу.
- k «лучших» союзников, т. е. $(r_M - k + 1), (r_M - k + 2), \dots, r_M$.
- $2k$ «лучших» противников, т. е. $l_B, (l_B + 1), \dots, (l_B + 2k - 1)$.

Нужно проверить, можно ли как-то попарить.

- Большие ограничения $\Rightarrow$ что-то простое.
- Необходимое условие: достаточная сумма, то есть

$$\sum_{z=r_M-k+1}^{r_M} (l_M + z) \geq \sum_{x=l_B}^{l_B+2k-1} x$$

H. Rigged Matchmaking

Строим пары $(l_M, z) - (x, y)$:

- Бинарный поиск по ответу.
- k «лучших» союзников, т. е. $(r_M - k + 1), (r_M - k + 2), \dots, r_M$.
- $2k$ «лучших» противников, т. е. $l_B, (l_B + 1), \dots, (l_B + 2k - 1)$.

Нужно проверить, можно ли как-то попарить.

- Большие ограничения $\Rightarrow$ что-то простое.
- Необходимое условие: достаточная сумма, то есть

$$\sum_{z=r_M-k+1}^{r_M} (l_M + z) \geq \sum_{x=l_B}^{l_B+2k-1} x$$

- Оказывается **достаточным** условием.

H. Rigged Matchmaking

Строим пары $(l_M, z) - (x, y)$:

- Бинарный поиск по ответу.
- k «лучших» союзников, т. е. $(r_M - k + 1), (r_M - k + 2), \dots, r_M$.
- $2k$ «лучших» противников, т. е. $l_B, (l_B + 1), \dots, (l_B + 2k - 1)$.

Нужно проверить, можно ли как-то попарить.

- Большие ограничения $\Rightarrow$ что-то простое.
- Необходимое условие: достаточная сумма, то есть

$$\sum_{z=r_M-k+1}^{r_M} (l_M + z) \geq \sum_{x=l_B}^{l_B+2k-1} x$$

- Оказывается **достаточным** условием.
- Проверка за $O(1) \Rightarrow$ решение за $O(\log(r_M + r_B))$.

H. Rigged Matchmaking

Конструкция в общем случае:

H. Rigged Matchmaking

Конструкция в общем случае:

- Берем четные позиции из левой половины и парим с первой четвертью правой половины.

H. Rigged Matchmaking

Конструкция в общем случае:

- Берем четные позиции из левой половины и парим с первой четвертью правой половины.
- Берем нечетные позиции слева и парим их со второй четвертью правой половины.

H. Rigged Matchmaking

Конструкция в общем случае:

- Берем четные позиции из левой половины и парим с первой четвертью правой половины.
- Берем нечетные позиции слева и парим их со второй четвертью правой половины.
- Получаем массив из k элементов, возрастающих на единицу. (Для четных k даже есть запас).

H. Rigged Matchmaking

Конструкция в общем случае:

- Берем четные позиции из левой половины и парим с первой четвертью правой половины.
- Берем нечетные позиции слева и парим их со второй четвертью правой половины.
- Получаем массив из k элементов, возрастающих на единицу. (Для четных k даже есть запас).

Пример: числа 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

H. Rigged Matchmaking

Конструкция в общем случае:

- Берем четные позиции из левой половины и парим с первой четвертью правой половины.
- Берем нечетные позиции слева и парим их со второй четвертью правой половины.
- Получаем массив из k элементов, возрастающих на единицу. (Для четных k даже есть запас).

Пример: числа 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

- $[2, 4]$ соединяем с $[7, 6] \Rightarrow [2 + 7, 4 + 6] = [9, 10]$.

H. Rigged Matchmaking

Конструкция в общем случае:

- Берем четные позиции из левой половины и парим с первой четвертью правой половины.
- Берем нечетные позиции слева и парим их со второй четвертью правой половины.
- Получаем массив из k элементов, возрастающих на единицу. (Для четных k даже есть запас).

Пример: числа 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

- $[2, 4]$ соединяем с $[7, 6] \Rightarrow [2 + 7, 4 + 6] = [9, 10]$.
- $[1, 3, 5]$ соединяем с $[10, 9, 8] \Rightarrow [1 + 10, 3 + 9, 5 + 8] = [11, 12, 13]$.

H. Rigged Matchmaking

Конструкция в общем случае:

- Берем четные позиции из левой половины и парим с первой четвертью правой половины.
- Берем нечетные позиции слева и парим их со второй четвертью правой половины.
- Получаем массив из k элементов, возрастающих на единицу. (Для четных k даже есть запас).

Пример: числа 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

- $[2, 4]$ соединяем с $[7, 6] \Rightarrow [2 + 7, 4 + 6] = [9, 10]$.
- $[1, 3, 5]$ соединяем с $[10, 9, 8] \Rightarrow [1 + 10, 3 + 9, 5 + 8] = [11, 12, 13]$.
- Получили $[9, 10, 11, 12, 13]$.

I. Remove Colors

Удалить все вершины из дерева так, чтобы в каждый момент времени все компоненты были разноцветные, а суммарная стоимость минимальна

I. Remove Colors

Удалить все вершины из дерева так, чтобы в каждый момент времени все компоненты были разноцветные, а суммарная стоимость минимальна

- Построим ориентированный граф, задающий порядок удаление цветов

I. Remove Colors

Удалить все вершины из дерева так, чтобы в каждый момент времени все компоненты были разноцветные, а суммарная стоимость минимальна

- Построим ориентированный граф, задающий порядок удаление цветов
- $x \rightarrow y$ значит, что цвет x надо удалить перед цветом y

I. Remove Colors

Удалить все вершины из дерева так, чтобы в каждый момент времени все компоненты были разноцветные, а суммарная стоимость минимальна

- Построим ориентированный граф, задающий порядок удаление цветов
- $x \rightarrow y$ значит, что цвет x надо удалить перед цветом y
- Тогда ответ на задачу можно вычислить из конденсации данного графа

I. Remove Colors

- Зафиксируем цвет x

I. Remove Colors

- Зафиксируем цвет x
- Ребра надо кинуть во все такие цвета, что вершины этих цветов есть внутри минимального связного подграфа, включающего все вершины цвета x

I. Remove Colors

- Зафиксируем цвет x
- Ребра надо кинуть во все такие цвета, что вершины этих цветов есть внутри минимального связного подграфа, включающего все вершины цвета x
- Найдем корень этого подграфа — LCA всех вершин цвета x

I. Remove Colors

- Зафиксируем цвет x
- Ребра надо кинуть во все такие цвета, что вершины этих цветов есть внутри минимального связного подграфа, включающего все вершины цвета x
- Найдем корень этого подграфа — LCA всех вершин цвета x
- Тогда ребра надо кинуть во все вершины на вертикальных путях от вершин цвета x до LCA

I. Remove Colors

- Зафиксируем цвет x
- Ребра надо кинуть во все такие цвета, что вершины этих цветов есть внутри минимального связного подграфа, включающего все вершины цвета x
- Найдем корень этого подграфа — LCA всех вершин цвета x
- Тогда ребра надо кинуть во все вершины на вертикальных путях от вершин цвета x до LCA
- Пока что ребер $O(n^2)$, надо оптимизировать

I. Remove Colors

- Маленькое ограничение по памяти значит, что требуется решение с $O(n)$ памяти

I. Remove Colors

- Маленькое ограничение по памяти значит, что требуется решение с $O(n)$ памяти
- Забьем на это и придумаем решение за $O(n \log n)$

I. Remove Colors

- Маленькое ограничение по памяти значит, что требуется решение с $O(n)$ памяти
- Забьем на это и придумаем решение за $O(n \log n)$
- Построим двоичные подъемы и используем их, как дополнительные вершины графа

I. Remove Colors

- Маленькое ограничение по памяти значит, что требуется решение с $O(n)$ памяти
- Забьем на это и придумаем решение за $O(n \log n)$
- Построим двоичные подъемы и используем их, как дополнительные вершины графа
- Разобьем вертикальный путь на $O(\log n)$ частей по двоичным подъемам и кинем ребра в новые виртуальные вершины

I. Remove Colors

- Маленькое ограничение по памяти значит, что требуется решение с $O(n)$ памяти
- Забьем на это и придумаем решение за $O(n \log n)$
- Построим двоичные подъемы и используем их, как дополнительные вершины графа
- Разобьем вертикальный путь на $O(\log n)$ частей по двоичным подъемам и кинем ребра в новые виртуальные вершины
- Кинем ребра из подъема $(v, i + 1)$ в подъемы (v, i) и $(up_{v,i}, i)$

I. Remove Colors

- Маленькое ограничение по памяти значит, что требуется решение с $O(n)$ памяти
- Забьем на это и придумаем решение за $O(n \log n)$
- Построим двоичные подъемы и используем их, как дополнительные вершины графа
- Разобьем вертикальный путь на $O(\log n)$ частей по двоичным подъемам и кинем ребра в новые виртуальные вершины
- Кинем ребра из подъема $(v, i + 1)$ в подъемы (v, i) и $(up_{v,i}, i)$
- Кинем ребра из подъема $(v, 0)$ в v

I. Remove Colors

- Маленькое ограничение по памяти значит, что требуется решение с $O(n)$ памяти
- Забьем на это и придумаем решение за $O(n \log n)$
- Построим двоичные подъемы и используем их, как дополнительные вершины графа
- Разобьем вертикальный путь на $O(\log n)$ частей по двоичным подъемам и кинем ребра в новые виртуальные вершины
- Кинем ребра из подъема $(v, i + 1)$ в подъемы (v, i) и $(up_{v,i}, i)$
- Кинем ребра из подъема $(v, 0)$ в v
- Получилось решение за $O(n \log n)$ времени и памяти

I. Remove Colors

- Попробуем вместо этого написать алгоритм Косараю неявно

I. Remove Colors

- Попробуем вместо этого написать алгоритм Косараю неявно
- Напомним алгоритм: dfs по графу, чтобы найти топсорту, затем dfs по транспонированному графу для выделения КСС

I. Remove Colors

dfs по прямому графу

I. Remove Colors

dfs по прямому графу

- Используем те же вертикальные пути

I. Remove Colors

dfs по прямому графу

- Используем те же вертикальные пути
- Для каждой вершины поддерживаем указатель на самого раннего непосещенного предка

I. Remove Colors

dfs по прямому графу

- Используем те же вертикальные пути
- Для каждой вершины поддерживаем указатель на самого раннего непосещенного предка
- Пересчитываем лениво

I. Remove Colors

dfs по прямому графу

- Используем те же вертикальные пути
- Для каждой вершины поддерживаем указатель на самого раннего непосещенного предка
- Пересчитываем лениво
- Сжимаем пути

I. Remove Colors

dfs по транспонированному графу

I. Remove Colors

dfs по транспонированному графу

- Где находятся цвета, из которых ведут ребра в цвет y ?

I. Remove Colors

dfs по транспонированному графу

- Где находятся цвета, из которых ведут ребра в цвет u ?
- Зафиксируем вершину цвета u

I. Remove Colors

dfs по транспонированному графу

- Где находятся цвета, из которых ведут ребра в цвет y ?
- Зафиксируем вершину цвета y
- Тогда должна быть вершина цвета x в поддереве этой вершины, а корень подграфа цвета x должен быть ее предком

I. Remove Colors

dfs по транспонированному графу

- Где находятся цвета, из которых ведут ребра в цвет y ?
- Зафиксируем вершину цвета y
- Тогда должна быть вершина цвета x в поддереве этой вершины, а корень подграфа цвета x должен быть ее предком
- Будем поддерживать дерево отрезков над эйлеровым обходом

I. Remove Colors

dfs по транспонированному графу

- Где находятся цвета, из которых ведут ребра в цвет y ?
- Зафиксируем вершину цвета y
- Тогда должна быть вершина цвета x в поддереве этой вершины, а корень подграфа цвета x должен быть ее предком
- Будем поддерживать дерево отрезков над эйлеровым обходом
- В вершине будем хранить высоту корня подграфа соответствующего цвета

I. Remove Colors

dfs по транспонированному графу

- Где находятся цвета, из которых ведут ребра в цвет y ?
- Зафиксируем вершину цвета y
- Тогда должна быть вершина цвета x в поддереве этой вершины, а корень подграфа цвета x должен быть ее предком
- Будем поддерживать дерево отрезков над эйлеровым обходом
- В вершине будем хранить высоту корня подграфа соответствующего цвета
- Если минимум в поддереве зафиксированной вершины меньше глубины самой вершины, то пойдем dfs в этот цвет
- Когда посещаем цвет, обнуляем вершины дерева отрезков

I. Remove Colors

dfs по транспонированному графу

- Где находятся цвета, из которых ведут ребра в цвет y ?
- Зафиксируем вершину цвета y
- Тогда должна быть вершина цвета x в поддереве этой вершины, а корень подграфа цвета x должен быть ее предком
- Будем поддерживать дерево отрезков над эйлеровым обходом
- В вершине будем хранить высоту корня подграфа соответствующего цвета
- Если минимум в поддереве зафиксированной вершины меньше глубины самой вершины, то пойдем dfs в этот цвет
- Когда посещаем цвет, обнуляем вершины дерева отрезков
- Асимптотика решения: $O(n)$ памяти, $O(n \log n)$ времени

J. Shift the Number

Дано число n ; посчитать количество таких положительных целых x , что циклический сдвиг n на x вправо дает $n + x$

J. Shift the Number

Дано число n ; посчитать количество таких положительных целых x , что циклический сдвиг n на x вправо дает $n + x$

- Перебирать x слишком долго. Давайте вместо этого конвертируем n в строку и переберем, на сколько символов она циклически сдвинется, чтобы получить кандидата на $n + x$

J. Shift the Number

Дано число n ; посчитать количество таких положительных целых x , что циклический сдвиг n на x вправо дает $n + x$

- Перебирать x слишком долго. Давайте вместо этого конвертируем n в строку и переберем, на сколько символов она циклически сдвинется, чтобы получить кандидата на $n + x$
- Таких кандидатов только 9

J. Shift the Number

Дано число n ; посчитать количество таких положительных целых x , что циклический сдвиг n на x вправо дает $n + x$

- Перебирать x слишком долго. Давайте вместо этого конвертируем n в строку и переберем, на сколько символов она циклически сдвинется, чтобы получить кандидата на $n + x$
- Таких кандидатов только 9
- Пусть текущий кандидат n' ; давайте проверим $x = n' - n$

J. Shift the Number

Дано число n ; посчитать количество таких положительных целых x , что циклический сдвиг n на x вправо дает $n + x$

- Перебирать x слишком долго. Давайте вместо этого конвертируем n в строку и переберем, на сколько символов она циклически сдвинется, чтобы получить кандидата на $n + x$
- Таких кандидатов только 9
- Пусть текущий кандидат n' ; давайте проверим $x = n' - n$
- x должен быть неотрицательным; кроме того, циклический сдвиг на x должен давать в точности n'

J. Shift the Number

Дано число n ; посчитать количество таких положительных целых x , что циклический сдвиг n на x вправо дает $n + x$

- Перебирать x слишком долго. Давайте вместо этого конвертируем n в строку и переберем, на сколько символов она циклически сдвинется, чтобы получить кандидата на $n + x$
- Таких кандидатов только 9
- Пусть текущий кандидат n' ; давайте проверим $x = n' - n$
- x должен быть неотрицательным; кроме того, циклический сдвиг на x должен давать в точности n'
- Чтобы не сдвигать на большой x , можно сдвинуть на $x \bmod k$, где k — длина числа

J. Shift the Number

Дано число n ; посчитать количество таких положительных целых x , что циклический сдвиг n на x вправо дает $n + x$

- Перебирать x слишком долго. Давайте вместо этого конвертируем n в строку и переберем, на сколько символов она циклически сдвинется, чтобы получить кандидата на $n + x$
- Таких кандидатов только 9
- Пусть текущий кандидат n' ; давайте проверим $x = n' - n$
- x должен быть неотрицательным; кроме того, циклический сдвиг на x должен давать в точности n'
- Чтобы не сдвигать на большой x , можно сдвинуть на $x \bmod k$, где k — длина числа
- Аккуратно с отсортированным порядком!

K. Strange Array

Есть число n и m троек (l_i, r_i, x_i) . Постройте массив длины n , для которого $k \cdot \text{OR}_{i=1}^n a_i - \sum_{i=1}^m ((\text{XOR}_{j=l_i}^{r_i} a_j) \text{ AND } x_i)$ максимальное

K. Strange Array

Есть число n и m троек (l_i, r_i, x_i) . Постройте массив длины n , для которого $k \cdot \text{OR}_{i=1}^n a_i - \sum_{i=1}^m ((\text{XOR}_{j=l_i}^{r_i} a_j) \text{ AND } x_i)$ максимальное

- Задачу можно решать по каждому биту независимо

K. Strange Array

Есть число n и m троек (l_i, r_i, x_i) . Постройте массив длины n , для которого $k \cdot \text{OR}_{i=1}^n a_i - \sum_{i=1}^m ((\text{XOR}_{j=l_i}^{r_i} a_j) \text{ AND } x_i)$ максимальное

- Задачу можно решать по каждому биту независимо
- Для бита i либо в каком-то числе в массиве есть 1, тогда побитовое ИЛИ не равно 0

K. Strange Array

Есть число n и m троек (l_i, r_i, x_i) . Постройте массив длины n , для которого $k \cdot \text{OR}_{i=1}^n a_i - \sum_{i=1}^m ((\text{XOR}_{j=l_i}^{r_i} a_j) \text{ AND } x_i)$ максимальное

- Задачу можно решать по каждому биту независимо
- Для бита i либо в каком-то числе в массиве есть 1, тогда побитовое ИЛИ не равно 0
- Иначе побитовое ИЛИ равно 0

K. Strange Array

Есть число n и m троек (l_i, r_i, x_i) . Постройте массив длины n , для которого $k \cdot \text{OR}_{i=1}^n a_i - \sum_{i=1}^m ((\text{XOR}_{j=l_i}^{r_i} a_j) \text{ AND } x_i)$ максимальное

- Задачу можно решать по каждому биту независимо
- Для бита i либо в каком-то числе в массиве есть 1, тогда побитовое ИЛИ не равно 0
- Иначе побитовое ИЛИ равно 0
- Элементы массива либо 0, либо 1; давайте проверим, можно ли какие-то из них сделать 1 так, чтобы суммарный штраф за отрезки был меньше k

K. Strange Array

Есть число n и m троек (l_i, r_i, x_i) . Постройте массив длины n , для которого $k \cdot \text{OR}_{i=1}^n a_i - \sum_{i=1}^m ((\text{XOR}_{j=l_i}^{r_i} a_j) \text{ AND } x_i)$ максимальное

- Задачу можно решать по каждому биту независимо
- Для бита i либо в каком-то числе в массиве есть 1, тогда побитовое ИЛИ не равно 0
- Иначе побитовое ИЛИ равно 0
- Элементы массива либо 0, либо 1; давайте проверим, можно ли какие-то из них сделать 1 так, чтобы суммарный штраф за отрезки был меньше k
- Если так нельзя, выгодно все выставить в этом бите по 0

K. Strange Array

Рассмотрим префиксные ксору: $p_i = \text{XOR}_{j=1}^i a_j$

K. Strange Array

Рассмотрим префиксные ксору: $p_i = \text{XOR}_{j=1}^i a_j$

- $p_0 = 0$, либо выгодно какие-то p_i сделать равными 1, либо нет

K. Strange Array

Рассмотрим префиксные ксору: $p_i = \text{XOR}_{j=1}^i a_j$

- $p_0 = 0$, либо выгодно какие-то p_i сделать равными 1, либо нет
- Благодаря префиксным ксорам для каждой тройки нас интересует $p_{l_i-1} \text{ XOR } p_{r_i}$

K. Strange Array

Рассмотрим префиксные ксору: $p_i = \text{XOR}_{j=1}^i a_j$

- $p_0 = 0$, либо выгодно какие-то p_i сделать равными 1, либо нет
- Благодаря префиксным ксорам для каждой тройки нас интересует $p_{l_i-1} \text{ XOR } p_{r_i}$
- Если в этой тройке соответствующий бит x_i равен 1, а $p_{l_i-1} \neq p_{r_i}$, к штрафу добавляется 1

K. Strange Array

Рассмотрим префиксные ксору: $p_i = \text{XOR}_{j=1}^i a_j$

- $p_0 = 0$, либо выгодно какие-то p_i сделать равными 1, либо нет
- Благодаря префиксным ксорам для каждой тройки нас интересует $p_{l_i-1} \text{ XOR } p_{r_i}$
- Если в этой тройке соответствующий бит x_i равен 1, а $p_{l_i-1} \neq p_{r_i}$, к штрафу добавляется 1
- Построим неориентированный граф из $n + 1$ вершин, где каждая вершина соответствует префиксному ксору

K. Strange Array

Рассмотрим префиксные ксору: $p_i = \text{XOR}_{j=1}^i a_j$

- $p_0 = 0$, либо выгодно какие-то p_i сделать равными 1, либо нет
- Благодаря префиксным ксорам для каждой тройки нас интересует $p_{l_i-1} \text{ XOR } p_{r_i}$
- Если в этой тройке соответствующий бит x_i равен 1, а $p_{l_i-1} \neq p_{r_i}$, к штрафу добавляется 1
- Построим неориентированный граф из $n + 1$ вершин, где каждая вершина соответствует префиксному ксору
- Ребро между двумя вершинами означает, что если в них поставить разные числа, то штраф увеличивается на 1

K. Strange Array

Дан неориентированный граф, надо разбить вершины на два множества так, чтобы кол-во ребер, соединяющих разные множества, было как можно меньше

K. Strange Array

Дан неориентированный граф, надо разбить вершины на два множества так, чтобы кол-во ребер, соединяющих разные множества, было как можно меньше

- Это задача о глобальном минимальном разрезе

K. Strange Array

Дан неориентированный граф, надо разбить вершины на два множества так, чтобы кол-во ребер, соединяющих разные множества, было как можно меньше

- Это задача о глобальном минимальном разрезе
- Есть алгоритм Штор-Вагнера за $O(V^3)$

K. Strange Array

Дан неориентированный граф, надо разбить вершины на два множества так, чтобы кол-во ребер, соединяющих разные множества, было как можно меньше

- Это задача о глобальном минимальном разрезе
- Есть алгоритм Штор-Вагнера за $O(V^3)$
- Если вы его не знаете, можно решать потоками: переберем вершину от 1 до n , которая будет в множестве 1, и найдем минразрез между 0 и этой вершиной

K. Strange Array

Дан неориентированный граф, надо разбить вершины на два множества так, чтобы кол-во ребер, соединяющих разные множества, было как можно меньше

- Это задача о глобальном минимальном разрезе
- Есть алгоритм Штор-Вагнера за $O(V^3)$
- Если вы его не знаете, можно решать потоками: переберем вершину от 1 до n , которая будет в множестве 1, и найдем минразрез между 0 и этой вершиной
- Минразрез равен максимальному потоку

K. Strange Array

Дан неориентированный граф, надо разбить вершины на два множества так, чтобы кол-во ребер, соединяющих разные множества, было как можно меньше

- Это задача о глобальном минимальном разрезе
- Есть алгоритм Штор-Вагнера за $O(V^3)$
- Если вы его не знаете, можно решать потоками: переберем вершину от 1 до n , которая будет в множестве 1, и найдем минразрез между 0 и этой вершиной
- Минразрез равен максимальному потоку
- Даже самый наивный алгоритм поиска макс. потока даст асимптотику $O(E^2)$ на поиск глобального минимального разреза

L. Red and Blue Edges

Дан неориентированный граф, в него добавляются ребра двух цветов. После каждого запроса надо считать, какое максимальное кол-во ребер можно удалить, чтобы связность не нарушилось, а кол-во удаленных ребер обоих цветов было равно

L. Red and Blue Edges

Дан неориентированный граф, в него добавляются ребра двух цветов. После каждого запроса надо считать, какое максимальное кол-во ребер можно удалить, чтобы связность не нарушилось, а кол-во удаленных ребер обоих цветов было равно

- Для начала поймем, как решать медленно

L. Red and Blue Edges

Дан неориентированный граф, в него добавляются ребра двух цветов. После каждого запроса надо считать, какое максимальное кол-во ребер можно удалить, чтобы связность не нарушилось, а кол-во удаленных ребер обоих цветов было равно

- Для начала поймем, как решать медленно
- Как понять, что можно удалить k красных ребер и k синих?

L. Red and Blue Edges

Дан неориентированный граф, в него добавляются ребра двух цветов. После каждого запроса надо считать, какое максимальное кол-во ребер можно удалить, чтобы связность не нарушилось, а кол-во удаленных ребер обоих цветов было равно

- Для начала поймем, как решать медленно
- Как понять, что можно удалить k красных ребер и k синих?
- Если текущее кол-во компонент равно s , то должно остаться хотя бы $n - s$ ребер

L. Red and Blue Edges

Дан неориентированный граф, в него добавляются ребра двух цветов. После каждого запроса надо считать, какое максимальное кол-во ребер можно удалить, чтобы связность не нарушилось, а кол-во удаленных ребер обоих цветов было равно

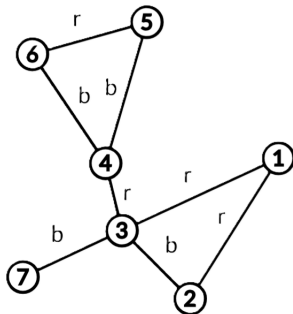
- Для начала поймем, как решать медленно
- Как понять, что можно удалить k красных ребер и k синих?
- Если текущее кол-во компонент равно c , то должно остаться хотя бы $n - c$ ребер
- Если $m - 2 \cdot k < n - c$, точно нельзя

L. Red and Blue Edges

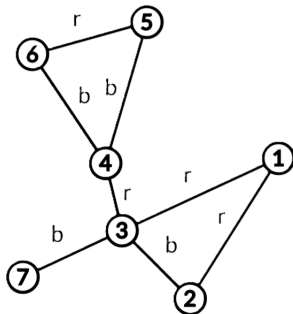
Дан неориентированный граф, в него добавляются ребра двух цветов. После каждого запроса надо считать, какое максимальное кол-во ребер можно удалить, чтобы связность не нарушилось, а кол-во удаленных ребер обоих цветов было равно

- Для начала поймем, как решать медленно
- Как понять, что можно удалить k красных ребер и k синих?
- Если текущее кол-во компонент равно c , то должно остаться хотя бы $n - c$ ребер
- Если $m - 2 \cdot k < n - c$, точно нельзя
- В противном случае можно не всегда

L. Red and Blue Edges

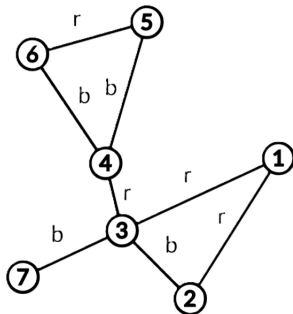


L. Red and Blue Edges



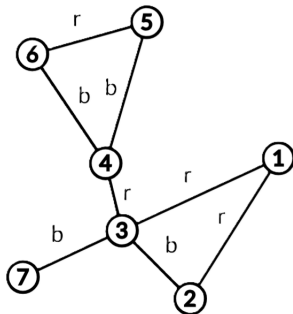
- 4 — 5 — 6: хотя бы одно синее

L. Red and Blue Edges



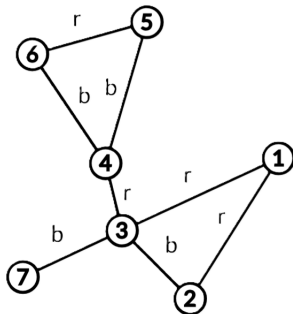
- 4 — 5 — 6: хотя бы одно синее
- 3 — 4: хотя бы одно красное

L. Red and Blue Edges



- 4 — 5 — 6: хотя бы одно синее
- 3 — 4: хотя бы одно красное
- 1 — 2 — 3: хотя бы одно красное

L. Red and Blue Edges



- 4 — 5 — 6: хотя бы одно синее
- 3 — 4: хотя бы одно красное
- 1 — 2 — 3: хотя бы одно красное
- 3 — 7: хотя бы одно синее

L. Red and Blue Edges

Можно найти B_{min} и R_{min} — минимальное кол-во ребер синего и красного цвета, которое должно быть

L. Red and Blue Edges

Можно найти B_{min} и R_{min} — минимальное кол-во ребер синего и красного цвета, которое должно быть

- Для нахождения B_{min} можно в DSU добавлять сначала все красные ребра, потом все синие

L. Red and Blue Edges

Можно найти B_{min} и R_{min} — минимальное кол-во ребер синего и красного цвета, которое должно быть

- Для нахождения B_{min} можно в DSU добавлять сначала все красные ребра, потом все синие
- Для R_{min} наоборот

L. Red and Blue Edges

Можно найти B_{min} и R_{min} — минимальное кол-во ребер синего и красного цвета, которое должно быть

- Для нахождения B_{min} можно в DSU добавлять сначала все красные ребра, потом все синие
- Для R_{min} наоборот
- Любое кол-во синих и красных ребер b и r , такое, что $b \geq B_{min}$, $r \geq R_{min}$ и $b + r \geq n - c$, нам доступно

L. Red and Blue Edges

Можно найти B_{min} и R_{min} — минимальное кол-во ребер синего и красного цвета, которое должно быть

- Для нахождения B_{min} можно в DSU добавлять сначала все красные ребра, потом все синие
- Для R_{min} наоборот
- Любое кол-во синих и красных ребер b и r , такое, что $b \geq B_{min}$, $r \geq R_{min}$ и $b + r \geq n - c$, нам доступно
- Можно найти максимальное k , для которого число красных ребер не станет меньше R_{min} , а синих не станет меньше B_{min}

Как обрабатывать запросы?

Как обрабатывать запросы?

- Надо поддерживать R_{min} и B_{min} , а также кол-во компонент с

Как обрабатывать запросы?

- Надо поддерживать R_{min} и B_{min} , а также кол-во компонент c
- Для того, чтобы узнать c , достаточно DSU

Как обрабатывать запросы?

- Надо поддерживать R_{min} и B_{min} , а также кол-во компонент c
- Для того, чтобы узнать c , достаточно DSU
- Для B_{min} можно поддерживать DSU на **красных** ребрах, и B_{min} равно тому числу, на которое отличается кол-во компонент в «красном» DSU от c

Как обрабатывать запросы?

- Надо поддерживать R_{min} и B_{min} , а также кол-во компонент c
- Для того, чтобы узнать c , достаточно DSU
- Для B_{min} можно поддерживать DSU на **красных** ребрах, и B_{min} равно тому числу, на которое отличается кол-во компонент в «красном» DSU от c
- Аналогично с R_{min}

Как обрабатывать запросы?

- Надо поддерживать R_{min} и B_{min} , а также кол-во компонент c
- Для того, чтобы узнать c , достаточно DSU
- Для B_{min} можно поддерживать DSU на **красных** ребрах, и B_{min} равно тому числу, на которое отличается кол-во компонент в «красном» DSU от c
- Аналогично с R_{min}
- Нужно поддерживать 3 DSU: общее, «красное» и «синее»

M. Tactical Game

Есть ряд из n клеток, задаваемый строкой из 0 и 1, где 0 — пустая клетка, 1 — противник. Можно за одно действие либо убить противника, либо выбрать противника и убить его и всех его соседей (но такое действие можно провести ровно один раз). Посчитать минимальное количество действий, чтобы убить всех

Есть ряд из n клеток, задаваемый строкой из 0 и 1, где 0 — пустая клетка, 1 — противник. Можно за одно действие либо убить противника, либо выбрать противника и убить его и всех его соседей (но такое действие можно провести ровно один раз). Посчитать минимальное количество действий, чтобы убить всех

- По умолчанию ответ равен количеству единиц в строке, если всех за одно действие убиваем

Есть ряд из n клеток, задаваемый строкой из 0 и 1, где 0 — пустая клетка, 1 — противник. Можно за одно действие либо убить противника, либо выбрать противника и убить его и всех его соседей (но такое действие можно провести ровно один раз). Посчитать минимальное количество действий, чтобы убить всех

- По умолчанию ответ равен количеству единиц в строке, если всех за одно действие убиваем
- Давайте попробуем понять, сколько действий можно сэкономить

Есть ряд из n клеток, задаваемый строкой из 0 и 1, где 0 — пустая клетка, 1 — противник. Можно за одно действие либо убить противника, либо выбрать противника и убить его и всех его соседей (но такое действие можно провести ровно один раз). Посчитать минимальное количество действий, чтобы убить всех

- По умолчанию ответ равен количеству единиц в строке, если всех за одно действие убиваем
- Давайте попробуем понять, сколько действий можно сэкономить
- Если есть 2 противника рядом, можно сэкономить одно действие, если 3 рядом — можно сэкономить два

Есть ряд из n клеток, задаваемый строкой из 0 и 1, где 0 — пустая клетка, 1 — противник. Можно за одно действие либо убить противника, либо выбрать противника и убить его и всех его соседей (но такое действие можно провести ровно один раз). Посчитать минимальное количество действий, чтобы убить всех

- По умолчанию ответ равен количеству единиц в строке, если всех за одно действие убиваем
- Давайте попробуем понять, сколько действий можно сэкономить
- Если есть 2 противника рядом, можно сэкономить одно действие, если 3 рядом — можно сэкономить два
- Самый простой способ реализации — проверить, есть ли в строке подстрока 11 или 111